BROWDY AND NEIMARK, P.L.L.C.
ATTORNEYS AT LAW
PATENT AND TRADEMARK CAUSES
SUITE 300
624 NINTH STREET, N.W.
WASHINGTON, D.C. 20001-5303
TELEPHONE (202)-628-5197

ALVIN BROWDY (1917-1998)
SHERIDAN NEIMARK
ROGER L. BROWDY

ANNE M. KORNBAU
NORMAN J. LATKER

OF COUNSEL
IVER P. COOPER

TELECOPIER FACSIMILE
(202) 737-3528
(202) 393-1012

E MAIL
mail@browdyneimark.com

PATENT AGENT
ALLEN C. YUN, PH D.

JC511 U.S. PTO
09/630411
08/01/00

August 1, 2000

JC803 U.S. PTO
08/01/00

Hon. Assistant Commissioner for Patents
Box Patent Appln
Washington, D.C. 20231

Re: New Patent Application in U.S.
Applicant: Eitan FARCHI et al.
Title: COMPUTER-IMPLEMENTED METHOD AND SYSTEM FOR AUTOMATICALLY...
Atty's Docket: FARCHI 1

Sir:

Attached herewith is the above-identified application for Letters Patent including:

[X] Specification (12 pages), claims (11 pages) and abstract (1 page)
[X] 4 Sheet Drawings (Figures 1-5)
  [X] Formal [ ] Informal
[X] Declaration and Power of Attorney (2 pages)
  [X] Newly executed [ ]Copy from prior application no. ___
[ ] Preliminary Amendment
  [ ] Computer-readable Sequence Listing
[ ] Supplemental Preliminary Amendment adding new claims -
[ ] Information Disclosure Statement with 1449 and references
[ ] A verified statement to establish small entity status under 37 CFR §1.9 and 37 CFR §1.27 (1 page)
[X] Please charge my American Express Account Form PTO-2038 attached in the amount of $ 1,506.00 to cover:
  [X] The filing fee calculated as follows (including any preliminary amendment for entry prior to calculation of the filing fee):

CLAIMS AS FILED				
FOR	NUMBER FILED	NUMBER EXTRA	RATE	BASIC FEE $ 690.00
TOTAL CLAIMS	35 - 20	= 15	x 18	$ 270.00
INDEPENDENT CLAIMS	10- 3	= 7	x 78	$ 546.00
[ ] Multiple Dependent Claim Presented			+ 260	--
[ ]Reduction of 1/2 for Small Entity				$
TOTAL FILING FEE				$1,506.00

[ ] Any additional fee required by the filing of an enclosed preliminary or supplemental preliminary amendment (for entry after calculation of the filing fee) has been calculated as shown below:

	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NO. PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE	CALCULATION
TOTAL		-	=	x 18	
INDEP.		-	=	x 78	
[ ] Multiple Dependent Claim Presented				+ 260	
[ ] Reduction by 1/2 for Small Entity					
Total Additional Fee =					

[ ] Other Fees: ______________________.
[ ] Other Attachments: ____________________.
[X] Return Receipt Postcard (in duplicate)

The following statements are applicable:

[ ] The benefit under 35 USC §119 is claimed of the filing date of: Application No.______ in _________ on________. A certified copy of said priority document [ ] is attached [ ] was filed in progenitor case ___________ on _______________.

[ ] The present application is a Continuation Divisional Continuation-in-part of prior claims the benefit of U.S. Provisional application no. , filed .

[ ] Incorporation By Reference. The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied herewith, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

[ ] A signed statement deleting inventor(s) named in the prior application is attached.

[ ] The prior application was assigned to: ______________________________

[ ] Amend the specification by inserting before the first line the sentence:
--This is a continuation/division/continuation-in-part claims the benefit of U.S. Provisional of copending parent application Serial No. , filed .--

[ ] Certain documents were previously cited or submitted to the Patent and Trademark Office in the following prior application ___________, which is relied upon under 35 USC §120. Applicants identify these documents by attaching hereto a form PTO-1449 listing these documents, and request that they be considered and made of record in accordance with 37 CFR §1.98(d). Per Section 1.98(d), copies of these documents need not be filed in this application.

[ ] A verified statement claiming small entity status is enclosed in progenitor application no. , filed . Status is still proper and desired.

[ ] The undersigned attorney of record hereby revokes the powers of attorney of:

[ ] The undersigned attorney of record hereby appoints associate power of attorney, to prosecute this application and to transact all business in the Patent and Trademark Office in connection therewith to:

[X] The Commissioner is hereby authorized to charge payment of the following additional fees associated with this communication or credit any overpayments to Deposit Account No. 02-4035:
[X] Any additional filing fees required under 37 CFR §1.16.
[X] Any patent application processing fees under 37 CFR §1.17.

[X] The Commissioner is hereby authorized to charge payment of the following fees, based on any paper filed during the pendency of this application or any CPA thereof, to effect any amendment, petition, or other action requested in said paper or credit any overpayments to Deposit Account No. 02-4035:
[X] Any patent application processing fees under 37 CFR §1.17.
[ ] The issue fee set in 37 CFR §1.18 at or before mailing the Notice of Allowance, pursuant to 37 CFR §1.311(b).
[X] Any filing fees under 37 CFR §1.16 for presentation of extra claims.
[X] If a paper is untimely filed in this or any CPA thereof by Applicant(s), the Commissioner is hereby petitioned under 37 CFR. §1.136(a) for the minimum extension of time required to make said paper timely. In the event a petition for extension of time is made under the provisions of this paragraph, the Commissioner is hereby requested to charge any fee required under 37 CFR §1.17 to Deposit Account 02-4035.

[X] The Commissioner is hereby authorized to credit any overpayment of fees accompanying this paper to Deposit Account No. 02-4035.

Respectfully submitted,
BROWDY AND NEIMARK, P.L.L.C.

By: [illegible]
Sheridan Neimark
Registration No. 20,520

SN:wrd
f:\filing\newapplicaitiontransmittal\*.wpd

Computer-implemented method and system for automatically invoking a predetermined debugger command at a desired location of a single thread of a program

FIELD OF THE INVENTION

This invention relates to computer program debuggers.

BACKGROUND OF THE INVENTION

Program debugging is done in one of the following two ways. According to a first approach, a debugger is used, a debugger being a tool that enables a partial execution of the program, stopping at predefined points such as lines of code or variable values. A second way is log-based debugging, wherein print statements are added to the program. When a test is executed a log is created. This log is examined off-line by the programmer to look for places where the behavior of the program deviates from the expected behavior. There also exist debugging tools that display traces of executions and show what happens during the execution. Once the location of a problem is found using these tools, other tools such as a debugger are used to debug the program. It is not possible, after identifying the location of the problem, to return to the actual program to investigate the state of the program. Getting to the correct location using a debugger can be a difficult problem, because algorithmic debugging, i.e. locating a bug, is difficult once a fault occurs.

Algorithmic debugging is used to support locating a defect once a fault is found, as described in web site http://www.cs.nmsu.edu/~mikau/aadebug.html and "*Generalized algorithmic debugging and testing*" by Peter Fritzson *et al*. appearing in "*ACM Letters on programming languages and testing*", 1:303-322, 1992.

There are many existing debugging tools and there are several trace visualization tools available, but there are no tools that combine the use of debugging commands and trace control together.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide a method and system for automatically invoking a predetermined debugger command at a desired location of a single thread of a program.

To this end, there is provided in accordance with a first aspect of the invention a computer-implemented method for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread, said method comprising:

(i) embedding within said single thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded; and

(ii) running the program for reading said trace file and invoking said predetermined debugger command.

The method according to the invention thus combines log-based debugging and use of a debugger. By such means, it is possible to examine the log and then to start the program, using the debugger at a point selected by the user. Such an approach may be implemented on large, multi-threaded programs containing hundreds of threads.

Implementation is especially difficult in the context of multi-threaded programs, owing to the fact that execution may result in different interleaving each time the program is run. That is to say that the mutual sequence in which more than one thread perform their instructions may not be constant, since from each thread's point of view, all that is important is that it performs its own instructions in the correct sequence providing that the integrity of the values of variables as read by the thread is not compromised. Such integrity is assured by the replay mechanism as explained below. It does not matter if that sequence is interrupted to allow another thread to perform instructions. However, this may have an effect on correct generation of the trace file.

The solution according to the invention requires several components:

(א) an instrumentation scheme, which allows the user to put specialized print statements in the program, both manually and automatically, to create trace files.

(ב) a modified debugger that can be executed "against" the trace files. This debugger, when encountering an instrumentation statement, will check the trace file. If the trace file contains a debugger command (such as show current program status) it will execute it, else it will continue if appropriate. This is the appearance to the user. Implementation may be done without modifying the debugger.

(ג) a number of supporting algorithms used to re-execute the program with the same multi-threading interleaving, and, if needed, to create a naming scheme for threads or to match between threads and traces.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to understand the invention and to see how it may be carried out in practice, a preferred embodiment will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

Fig. 1 is a flow diagram showing the principal operating steps carried out by a method according to a first embodiment of the invention for invoking a debugger from a trace file;

Fig. 2 is a flow diagram showing the principal operating steps carried by a **"Trace-Print"** function inserted into program code in the method shown in Fig. 1;

Fig. 3 is a flow diagram showing the principal operating steps carried by a scheme for automatic naming of trace files as a function of a creating thread's name;

Fig. 4 is a pictorial representation of a tree structure depicting a scheme for automatic naming of trace files as a function of a creating thread's name; and

Fig. 5 is a block diagram showing functionally a system for carrying out the invention.

DETAILED DESCRIPTION OF THE INVENTION

Fig. 1 is a flow diagram showing the main operating steps carried out by a method according to a first embodiment of the invention for invoking a debugger from a trace file. In common with known tracing techniques, print statements are inserted into the program source code so as to output the value of a specified variable at each location in the program where the value of the variable is significant and where it may be important subsequently to invoke a debugger command. However, unlike known tracing techniques, the print statements that are inserted into the program source code are a modified print function,

"Trace-Print", that operates differently according to whether or not a trace file already exists. Specifically, the modified print function creates a trace file if the file does not exist, and writes the value of the variables thereto.

This having been done, a programmer takes the trace file and reviews it line by line. When he or she comes across a value which appears suspect, a debugger command is either inserted into the trace file before the line in the trace file corresponding to the suspect line or, alternatively substitutes the line containing suspect variable with a debugger command.

This having been done, the program is re-run. It is assumed that the program retraces its previous steps as recorded in the trace file until it reaches the first modified print function.

Fig. 2 shows the principal operating steps carried by the modified print function, **"Trace-Print"**. Thus, as noted above, on determining that the file does not exist, the **Trace-Print** function creates a trace file and writes the value of the variables thereto. However, if the trace file does already exist, then the current value of the variable output by the program at the current location of the **Trace-Print** function is compared with a respective line in the trace file. If they are different, the **Trace-Print** function construes the respective line in the trace file as a debugger command and invokes a debugger so as to execute the debugger command.

A particular example follows wherein different components of the invention are described in greater detail.

Instrumentation:

Functions of the form **Trace-Print** (Trace-Name, Content) are added to the program. Preferably, this is done automatically, using a menu, in all locations of a specific type defined by the programmer (e.g. beginning of routines, places where a particular variable "X" is used, etc.). In the case of multi-threaded programs, automatic choice of the trace granularity ensures replay ability. Alternatively, the **Trace-Print** functions may be added to the program manually. Replay ability is described in "*Deterministic Replay of Java Multithreaded Applications*", by Jong-Deok Choi and Harini Srinivasan, in the Proceedings of the ACM SIGMETRICS symposium on parallel and distributed tools, 1998.

When the program is executed on a test, a flag called "**compare**" is added. If **compare**=*false*, then the **Trace-Print** function prints Content into file **Trace-Name** in the form "**Trace: Content**".

If **compare**=*true*, then the program is executed under a modified debugger.

Modified Debugger:

Whenever a **Trace-Print** function is reached, if the content of Trace-Name is of the form "**Debug: Command**", then this debugger command is executed. As shown in Fig. 2, such execution can be performed either iteratively or in batch mode.

IF the content is of the form "**Trace: Content**", **THEN**

IF the content in the trace file equals **Content** of **Trace-Print**, then continue;

ELSE raise the debugger at this point with the appropriate error message.

Alternatively, the function **Trace-Print** is modified such that if the file **Trace-Name** is one of the debug-trace file names it either writes to the file as above or compares, much like the previous implementation but implemented within the program, possibly as a library. When a point is reached where the debugger is needed, it is invoked from within the program. This implementation will work only for debuggers that can be invoked while the program is running, as is the case with most debuggers. The advantage of this approach is that the debugger is not modified, and it is easy to port between debuggers.

Alternatively, a new application may be created that does not run under the debugger. Such an application accepts as input a trace file and a stream of outputs from the executing program and compares the stream to the trace file. If the stream is different, it halts the program and raises a debugger. Such an approach has the advantage that the user program is not modified and the application is not language specific. In either case, application of the debugger gives rise to delays and it is therefore hard to break the program exactly at the desired location.

One possible solution to this is to employ an architecture called "King" (or any other from the published solutions) for re-executing the program with the same interleaving as far as the **Trace-Print** function is concerned. The king is a synchronization object able to influence, record and enforce interleaving.

The king architecture is used to record the order in which concurrent programs are executed and to rerun concurrent programs in the same or similar order. This architecture is used to support interactive definition of timing related tests. The tester can select a thread and advance it to a chosen program location. A King architecture contains the following elements:

- The king architecture has two modes 'record' and 'replay'. When the program is run in record mode, the king gets called when a snapshot statement is reached. The king then records the order in which snapshot operations occurred. When the program is run in 'replay' mode, the king is called when snapshots statements are reached. The king then determines which thread to advance in order to force the recorded execution order.

- The king uses a language to record and replay an execution order. The simplest possible language used by the king is a sequence of commands each advancing a certain thread to the next snapshot instruction, e.g.,

thread 1 A; thread 1 B; thread 2 A;

- The user can use this language interactively to define the execution order.

When running multi-thread programs, each thread may write to its own trace file and when the programmer analyzes the trace file, it must clear to which thread the traced variables refer. To this end, some consistent naming scheme is required that ensures that allocates a unique name to each thread-dependent trace file and ensures that each thread writes to its own trace file.

In an example of the present invention, a method for identifying if there is no correlation between threads and traces uses bipartite matching. The method is based on the idea that initially each thread is matched to all traces. As a thread prints content, the matching is restricted to the subset of the traces that match that content. If there is no bipartite matching from the threads to the traces after a print, then the program is stopped so as to avoid executing debugger commands embedded in each of the traces at the wrong time.

Fig. 3 shows the principal steps associated with a consistent naming algorithm according to the invention for threads based on the following observations:

- A thread always has a parent thread (except for the Main Thread - i.e. the thread which starts the program).
- Since each thread executes its code sequentially, the creation time of every child thread, per parent thread, is unique.

Assumptions:

- When the system starts, there is only a single thread, called the Main Thread, or the Root Thread.
- There exists a thread bound index structure, i.e. for each thread created, there is a data structure, which holds an index counter.
- Update of the index counter is atomic.
- Thread creation time is defined as the first time a thread was created (e.g. using the new command for thread object creation in Java), rather than the time the new thread starts executing, which could be system dependent and inconsistent across program executions.

The algorithm:

As the Main Threads starts, it is named as Main, and an index structure is created for it, whose index is initialized to be 1. For each newly created child thread, at Thread creation time, do:

- Create child thread name by using its parent name as a prefix and its parent index as suffix.
- Increase parent thread's index by 1.
- Create an index structure for the child thread and initialize its index to 1.

Fig. 4 is a pictorial representation of a tree structure depicting the above-described algorithm for automatic naming of trace files as a function of a creating thread's name. Since by definition, thread creation time is unique, and since the creation time is used to produce a unique name for every newly created thread, every thread during the lifetime of a program has a unique name. In addition, this unique name is preserved across program re-execution, as long as each thread maintains the same order of child thread creation.

Fig. 5 is a block diagram showing functionally a computer-implemented system 10 for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread. The system 10 comprises a code modifier 11 for embedding within the program thread at the desired location thereof a utility which reads a trace file in which the predetermined debugger command has been previously embedded. A processor 12 reads the trace file during running of the program and invokes the predetermined debugger command. A file management system 13 is coupled to the processor 12 and is responsive to the embedded utility for checking whether a trace file exists, and for creating the trace file if it does not exist. A file modifier 14 coupled to the file management system 13 is responsive to the trace file being created for writing to trace file a traced value of at least one variable at the desired location of the program. A comparator 15 is coupled to the processor 12 for comparing a current value of the least one variable with a respective line in the trace file. If they are different, the comparator 15 construes the respective line in the trace file as a debugger command and invokes a debugger so as to execute the debugger command.

The program may be multi-threaded in which case there is further included a replay mechanism 16 coupled to the processor 12 for rerunning the program with identical interleaving as far as instrumentation statements are concerned. The processor 12 may further be adapted to run the program for reading a modified trace file readable by the program wherein at least one traced value is replaced or augmented by the debugger command.

Likewise, in the event that the system is used with multi-threaded program, the file management system 13 may be adapted to create for each thread a respective trace file having a name which is uniquely defined by a name of the respective thread, thereby allowing debugger commands embedded in any of the trace files to be executed during a respective one of the threads.

The file management system 13 may also be responsive to a predetermined execution-independent naming scheme for automatically naming the trace files. To this end, the file management system 13 includes an assignment unit 17 for assigning a root name to a root thread, and a thread-bound index structure 18 for holding for each thread a corresponding index counter, which is atomically incremented upon thread creation. The assignment unit 17 is responsive to creation of a child thread for assigning a name including a prefix indicative of a name of a respective parent thread and a suffix indicative of an index counter of the respective parent thread. The assignment unit 17 is responsive to no consistent naming being possible for attempting a bipartite matching between the threads and the traces such that every thread has a trace which contains what the thread printed, and for stopping the program so as to avoid executing debugger commands embedded in each of the traces at the wrong time if bipartite matching is not possible.

A bypass mechanism 18 coupled to the file modifier 14 allows creation of the trace file to be manually or automatically bypassed so that traces are created in respect of only a subset of the threads. The processor 12 may be adapted to read the modified trace file in respect of local views of the threads only, so as to avoid a need for synchronizing break-points in multiple threads of a multithreaded program.

It will also be understood that the system according to the invention may be a suitably programmed computer. Likewise, the invention contemplates a computer program being readable by a computer for executing the method of the invention. The invention further contemplates a machine-readable memory tangibly embodying a program of instructions executable by the machine for executing the method of the invention.

In the method claims that follow, alphabetic characters used to designate claim steps are provided for convenience only and do not imply any particular order of performing the steps.

CLAIMS:

1. A computer-implemented method for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread, said method comprising:

(a) embedding within said single thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded; and

(b) running the program for reading said trace file and invoking said predetermined debugger command.

2. The method according to Claim 35, wherein said utility performs the following steps:

i. checks whether a trace file exists,

ii. if the trace file does not exist, creates the trace file and writes thereto a traced value of at least one variable at said desired location of the program, and

iii. if the trace file does exist, compares a current value of the least one variable with a respective line in the trace file and if they are different construes the respective line in the trace file as a debugger command and invokes a debugger so as to execute the debugger command;

whereby running the program prior to carrying out said method traces one or more variables to the trace file, and step **(a)** includes modifying the trace file so as to replace or insert at least one traced value with the predetermined debugger command.

3. The method according to Claim 35, wherein the debugger attaches itself to a predetermined debugger command, which halts execution of the program and shows a state of the program at that time.

4. The method according to Claim 35, wherein the program is multi-threaded and there is included the further step of:

(c) providing a mechanism for rerunning the program with identical interleaving as far as instrumentation statements are concerned.

5. The method according to Claim 1, further including the step of creating the mechanism automatically using the instrumentation statements.

6. The method according to Claim 35, wherein the program includes multiple threads, each of which prints an invariant associated with a status of the program, said invariant having a value that remains constant regardless of the interleaving.

7. A computer-implemented method for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said method comprising:

(a) embedding within the specific thread of the program at said desired location thereof a utility which:

i) checks whether a trace file exists,

ii) if the trace file does not exist, creates the trace file and writes a traced value of at least one variable thereto at a desired location of the program, and

iii) if the trace file does exist, compares a current value of the least one variable with a respective line in the trace file and if they are different invokes a debugger so as to execute a debugger command embedded in the trace file in place of the traced value; and

(b) running the program for reading a modified trace file readable by the program wherein at least one traced value is replaced or augmented by said debugger command.

8. The method according to Claim 7, wherein the predetermined debugger command halts execution of the program and shows the state of the program at that time.

9. The method according to Claim 7, wherein the program is multi-threaded and there is included the further step of:

(c) providing a mechanism for rerunning the program with identical interleaving as far as instrumentation statements are concerned.

10. The method according to Claim 9, including the further step of creating the mechanism automatically using the instrumentation statements.

11. The method according to Claim 7, wherein the program includes multiple threads, each of which prints an invariant associated with a status of the program, said invariant having a value that remains constant regardless of the interleaving.

12. The method according to Claim 7, wherein the program includes multiple threads and step **(a)(ii)** includes:

i) creating for each thread a respective trace file having a name which is uniquely defined by a name of the respective thread;

thereby allowing debugger commands embedded in any of the trace files to be executed during a respective one of the threads.

13. The method according to Claim 12, further including the step of automatically naming said trace files according to a predetermined execution-independent naming scheme.

14. The method according to Claim 13, wherein said naming scheme includes the steps of:

i) assigning a root name to a root thread,

ii) maintaining a thread-bound index structure for holding for each thread a corresponding index counter which is atomically incremented upon thread creation, and

iii) upon creation of a child thread assigning a name including a prefix indicative of a name of a respective parent thread and a suffix indicative of an index counter of the respective parent thread.

15. The method according to Claim 11, wherein step **(a)(ii)** includes:

i) attempting a bipartite matching between the threads and the traces such that every thread has a trace which contains what the thread printed, and

ii) if said bipartite matching is not possible, then stopping the program so as to avoid executing debugger commands embedded in each of the traces at the wrong time.

16. The method according to Claim 11, further including providing a mechanism for manually or automatically bypassing step **(a)(ii)** so that traces are created in respect of only a subset of the threads.

17. The method according to Claim 11, wherein step **(b)** includes:

i) reading the modified trace file in respect of local views of the threads only, so as to avoid a need for synchronizing break-points in said multiple threads.

18. A computer program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread, said method comprising:

(a) embedding within said program thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded; and

(b) running the program for reading said trace file and invoking said predetermined debugger command.

19. A computer program product comprising a computer useable medium having computer readable program code embodied therein for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread, said computer program product comprising:

computer readable program code for causing the computer to embed within said program thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded; and

computer readable program code for causing the computer to run the program for reading said trace file and invoking said predetermined debugger command.

20. A computer program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said method comprising:

(a) embedding within the specific thread of the program at said desired location thereof a utility which:

i) checks whether a trace file exists,

ii) if the trace file does not exist, creates the trace file and writes a traced value of at least one variable thereto at a desired location of the program, and

iii) if the trace file does exist, compares a current value of the least one variable with a respective line in the trace file and if they are different invokes a debugger so as to execute a debugger command embedded in the trace file in place of the traced value; and

(b) running the program for reading a modified trace file readable by the program wherein at least one traced value is replaced or augmented by said debugger command.

21. A computer program product comprising a computer useable medium having computer readable program code embodied therein for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said computer program product comprising:

computer readable program code for causing the computer to embed within the specific thread of the program at said desired location thereof a utility which:

computer readable program code for causing the computer to checks whether a trace file exists,

computer readable program code for causing the computer to create the trace file if the trace file does not exist, and to write a traced value of at least one variable thereto at a desired location of the program, and

computer readable program code for causing the computer to compare a current value of the least one variable with a respective line in the trace file if the trace file does exist, and if they are different to invoke a debugger so as to execute a debugger command embedded in the trace file in place of the traced value; and

computer readable program code for causing the computer to run the program for reading a modified trace file readable by the program wherein at least one traced value is replaced or augmented by said debugger command.

22. A computer program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform method steps for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said method comprising:

(a) checking whether a trace file exists,

(b) if the trace file does not exist, creating the trace file and writes a traced value of at least one variable thereto at a desired location of the program, and

(c) if the trace file does exist, comparing a current value of said at least one variable with a respective line in the trace file and if they are different invoking a debugger so as to execute a debugger command embedded in the trace file in place of the traced value.

23. A computer program product comprising a computer useable medium having computer readable program code embodied therein for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said computer program product comprising:

computer readable program code for causing the computer to checks whether a trace file exists,

computer readable program code for causing the computer to create the trace file if the trace file does not exist, and to write a traced value of at least one variable thereto at a desired location of the program, and

computer readable program code for causing the computer to compare a current value of the at least one variable with a respective line in the trace file if the trace file does exist, and if they are different to invoke a debugger so as to execute a debugger command embedded in the trace file in place of the traced value.

24. A computer-implemented system for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread, said system comprising:

a code modifier for embedding within said single thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded, and

a processor for reading said trace file during running of the program and invoking said predetermined debugger command.

25. The system according to Claim 24, further including:

a file management system coupled to the processor and responsive to said utility for checking whether a trace file exists, and for creating the trace file if it does not exist,

a file modifier coupled to the file management system and responsive to the trace file being created for writing to trace file a traced value of at least one variable at said desired location of the program, and

a comparator coupled to the processor for comparing a current value of the least one variable with a respective line in the trace file and if they are different construing the respective line in the trace file as a debugger command and invoking a debugger so as to execute the debugger command.

26. The system according to Claim 24, wherein the debugger is adapted to attach itself to a predetermined debugger command, which halts execution of the program and shows a state of the program at that time.

27. The system according to Claim 24, wherein the program is multi-threaded and there is further included a replay mechanism coupled to the processor for rerunning the program with identical interleaving as far as instrumentation statements are concerned.

28. A computer-implemented system for automatically invoking a predetermined debugger function at a desired location of a specific thread of a program containing at least one thread, said system comprising:

a code modifier for embedding within said single thread at said desired location thereof a utility which reads a trace file in which said predetermined debugger command has been previously embedded,

a file management system coupled to the processor and responsive to said utility for checking whether a trace file exists, and for creating the trace file if it does not exist, a file modifier coupled to the file management system and responsive to the trace file being created for writing to trace file a traced value of at least one variable at said desired location of the program,

a processor for running the program for reading a modified trace file readable by the program wherein at least one traced value is replaced or augmented by said debugger command, and

a comparator coupled to the processor for comparing a current value of the least one variable with a respective line in the trace file and if they are different construing the respective line in the trace file as a debugger command and invoking a debugger so as to execute the debugger command.

29. The system according to Claim 28, wherein the program is multi-threaded and there is further included a replay mechanism coupled to the processor for rerunning the program with identical interleaving as far as instrumentation statements are concerned.

30. The system according to Claim 28, wherein the program includes multiple threads and the file management system is adapted to create for each thread a respective trace file having a name which is uniquely defined by a name of the respective thread, thereby allowing debugger commands embedded in any of the trace files to be executed during a respective one of the threads.

31. The system according to Claim 30, wherein the file management system is responsive to a predetermined execution-independent naming scheme for automatically naming said trace files.

32. The system according to Claim 31, wherein the file management system includes:

an assignment unit for assigning a root name to a root thread, and a thread-bound index structure for holding for each thread a corresponding index counter which is atomically incremented upon thread creation;

said assignment unit being responsive to creation of a child thread for assigning a name including a prefix indicative of a name of a respective parent thread and a suffix indicative of an index counter of the respective parent thread.

33. The system according to Claim 32, wherein the assignment unit is responsive to no consistent naming being possible for attempting a bipartite matching between the threads and the traces such that every thread has a trace which contains what the thread printed, and for stopping the program so as to avoid executing debugger commands embedded in each of the traces at the wrong time if said bipartite matching is not possible.

34. The system according to Claim 28, further including a bypass mechanism coupled to the file modifier for allowing creation of the trace file to be manually or automatically bypassed so that traces are created in respect of only a subset of the threads.

35. The system according to Claim 28, wherein the processor is adapted to read the modified trace file in respect of local views of the threads only, so as to avoid a need for synchronizing break-points in said multiple threads.

ABSTRACT

A computer-implemented method and system for automatically invoking a predetermined debugger command at a desired location of a single thread of a program containing at least one thread. At the desired location of the program thread, there is embedded a utility which reads a trace file in which the predetermined debugger command has been previously embedded. Upon re-running the program, the trace file is read and upon reaching the predetermined debugger command, the debugger attaches itself to the running process and executes the process from its current program counter. The debugger is invoked only if there is a discrepancy between successive runs of the program.

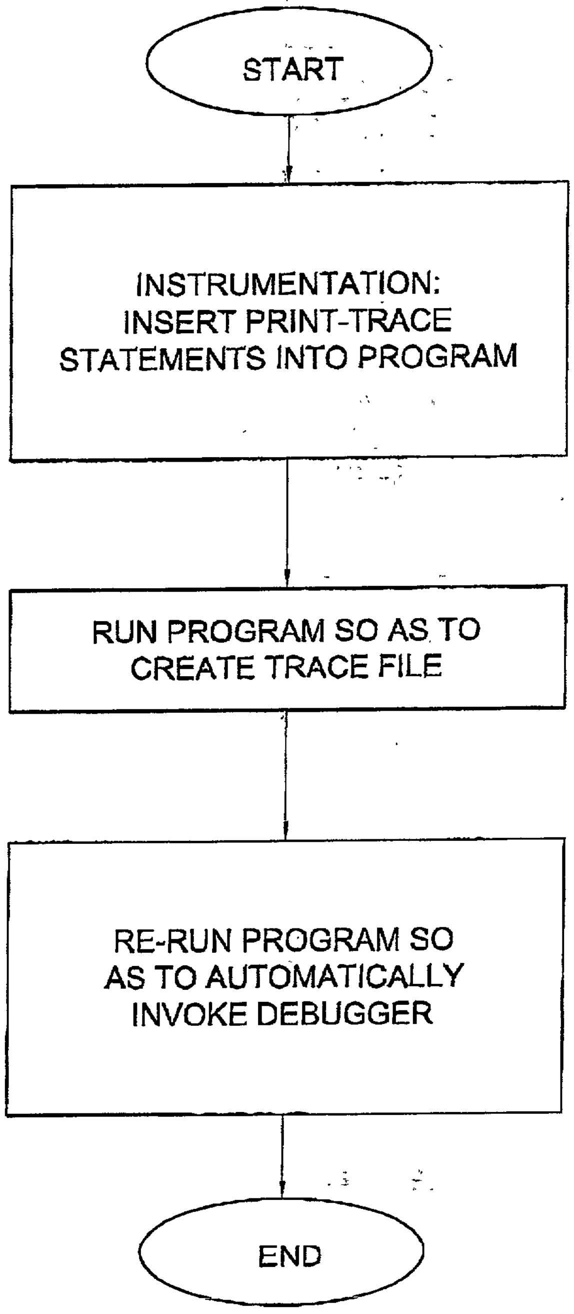
START
INSTRUMENTATION: INSERT PRINT-TRACE STATEMENTS INTO PROGRAM
RUN PROGRAM SO AS TO CREATE TRACE FILE
RE-RUN PROGRAM SO AS TO AUTOMATICALLY INVOKE DEBUGGER
END

FIG. 1

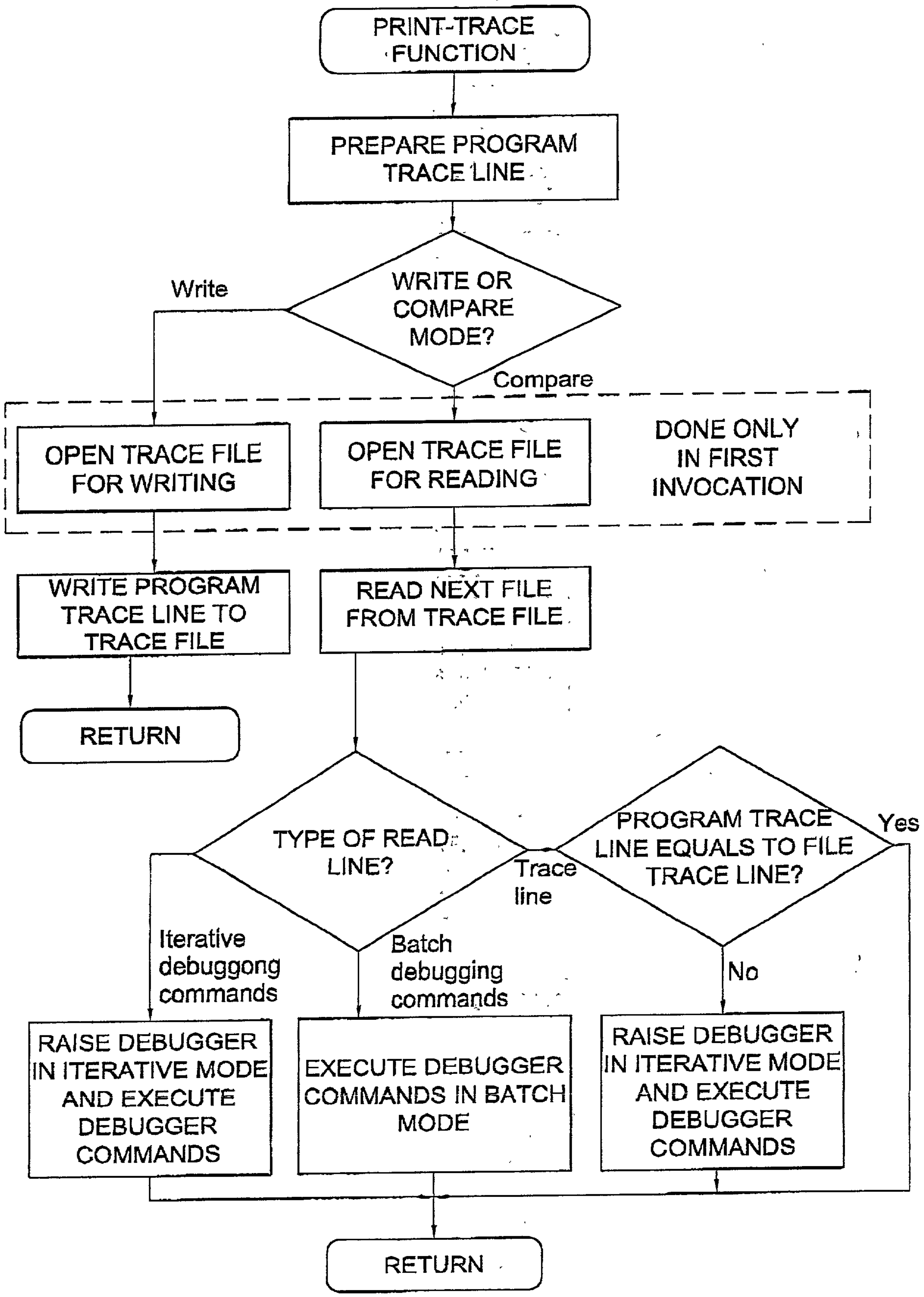
PRINT-TRACE FUNCTION
PREPARE PROGRAM TRACE LINE
WRITE OR COMPARE MODE?
Write
Compare
OPEN TRACE FILE FOR WRITING
OPEN TRACE FILE FOR READING
DONE ONLY IN FIRST INVOCATION
WRITE PROGRAM TRACE LINE TO TRACE FILE
READ NEXT FILE FROM TRACE FILE
RETURN
TYPE OF READ LINE?
Trace line
PROGRAM TRACE LINE EQUALS TO FILE TRACE LINE?
Yes
Iterative debuggong commands
Batch debugging commands
No
RAISE DEBUGGER IN ITERATIVE MODE AND EXECUTE DEBUGGER COMMANDS
EXECUTE DEBUGGER COMMANDS IN BATCH MODE
RAISE DEBUGGER IN ITERATIVE MODE AND EXECUTE DEBUGGER COMMANDS
RETURN

FIG. 2

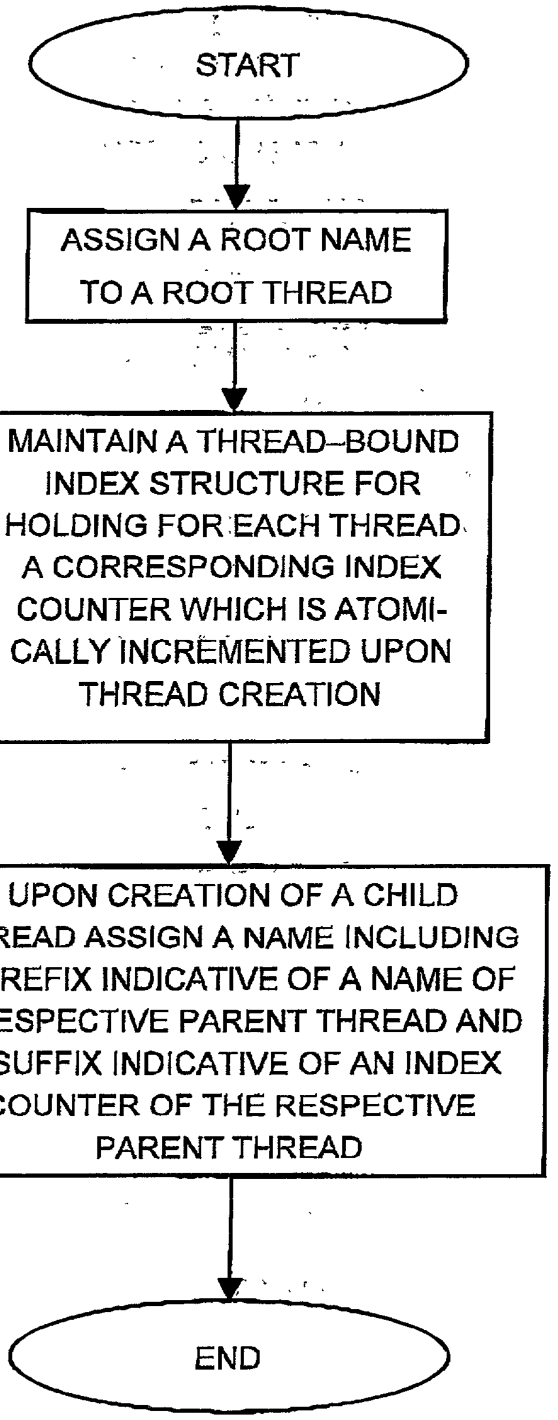
START
ASSIGN A ROOT NAME TO A ROOT THREAD
MAINTAIN A THREAD–BOUND INDEX STRUCTURE FOR HOLDING FOR EACH THREAD A CORRESPONDING INDEX COUNTER WHICH IS ATOMICALLY INCREMENTED UPON THREAD CREATION
UPON CREATION OF A CHILD THREAD ASSIGN A NAME INCLUDING A PREFIX INDICATIVE OF A NAME OF A RESPECTIVE PARENT THREAD AND A SUFFIX INDICATIVE OF AN INDEX COUNTER OF THE RESPECTIVE PARENT THREAD
END

FIG. 3

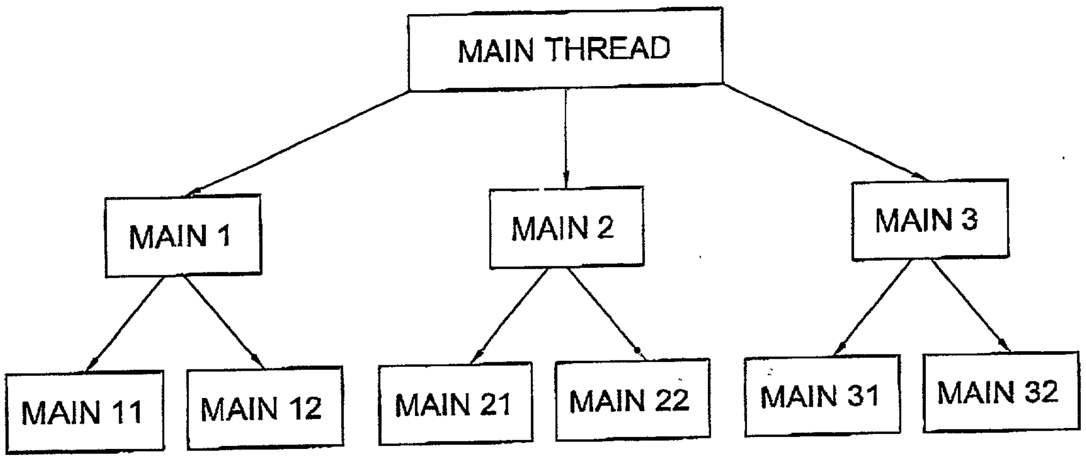
MAIN THREAD
MAIN 1
MAIN 2
MAIN 3
MAIN 11
MAIN 12
MAIN 21
MAIN 22
MAIN 31
MAIN 32

FIG. 4

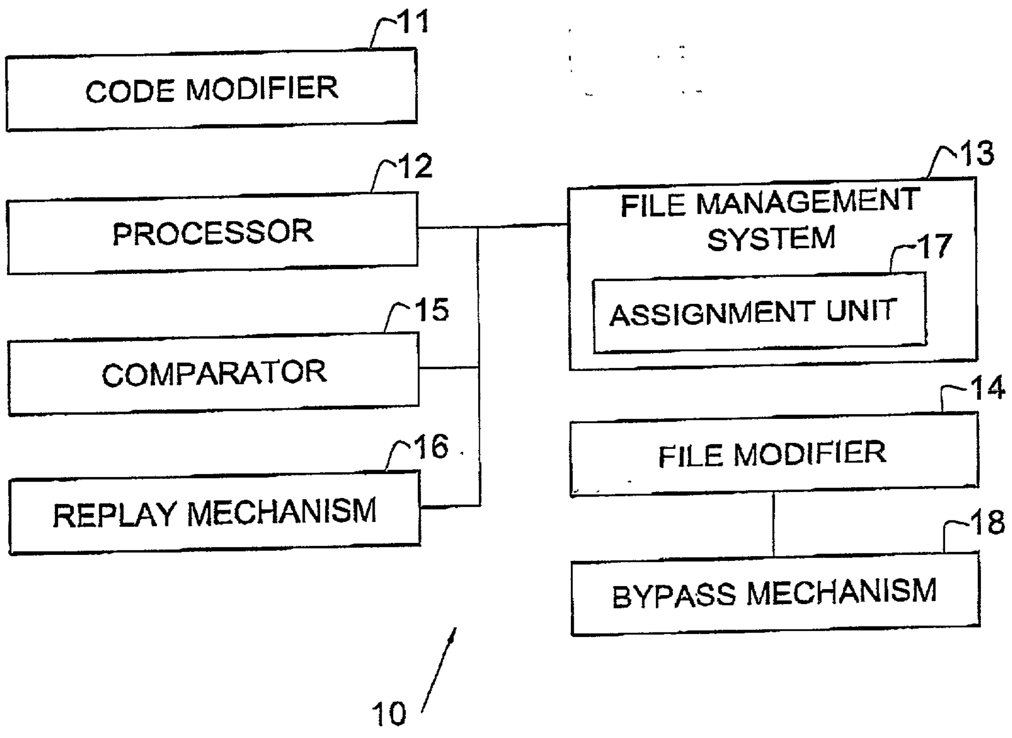
11
CODE MODIFIER
12
PROCESSOR
15
COMPARATOR
16
REPLAY MECHANISM
13
FILE MANAGEMENT SYSTEM
17
ASSIGNMENT UNIT
14
FILE MODIFIER
18
BYPASS MECHANISM
10

FIG. 5

116 949.9

Page 1 of 2 ☒ Original ☐ Supplemental Atty. Docket:

Combined Declaration for Patent Application and Power of Attorney

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; and that I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

<u>**COMPUTER-IMPLEMENTED METHOD AND SYSTEM FOR AUTOMATICALLY INVOKING A PREDETERMINED DEBUGGER COMMAND AT A DESIRED LOCATION OF A SINGLE THREAD OF A PROGRAM**</u>

the specification of which (check one)

- ☒ is attached hereto
- ☐ was filed in the United States under 35 U.S.C. §111 on ________________, as USSN ______________*; or
- ☐ was/will be filed in the U.S. under 35 U.S.C. §371 by entry into the U.S. national stage of an international (PCT) application, PCT/__________; filed __________, entry requested on __________*; national stage application received USSN __________*; §371/§102(e) date __________* (* if known),

and was amended on ______________________________ (if applicable)

(Include dates of amendments under PCT Art. 19 and 34 if PCT)

I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above; and I acknowledge the duty to disclose to the Patent and Trademark Office (PTO) all information known by me to be material to patentability as defined in 37 C.F.R. § 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. §§ 119, 365 of any prior foreign application(s) for patent or inventor's certificate, or prior PCT application(s) designating a country other than the U.S., listed below with the "Yes" box checked and have also identified below any such application having a filing date before that of the application on which priority is claimed:

(Number)	(Country)	(Day Month Year Filed)	☐ YES	☐ NO
(Number)	(Country)	(Day Month Year Filed)	☐ YES	☐ NO

I hereby claim the benefit under 35 U.S.C. § 120 of any prior U.S. non-provisional Application(s) or prior PCT Application(s) designating the U.S. listed below, or under § 119(e) of any prior U.S. provisional applications listed below, and insofar as the subject matter of each of the claims of this application is not disclosed in such U.S. or PCT application in the manner provided by the first paragraph of U.S.C. §112, I acknowledge the duty to disclose to the PTO all information as defined in 37 C.F.R. §1.56(a) which occurred between the filing date of the prior application and the national filing date of this application:

(Application Serial No.)	(Day Month Year Filed)	(Status : patented, pending, abandoned)
(Application Serial No.)	(Day Month Year Filed)	(Status : patented, pending, abandoned)
(Application Serial No.)	(Day Month Year Filed)	(Status : patented, pending, abandoned)

I hereby appoint the following attorneys, with full power of substitution, association, and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith.

SHERIDAN NEIMARK,	REG. NO. 20,520	- ROGER L. BROWDY,	REG. NO. 25,618	- ANNE M. KORNBAU,	REG. NO. 25,884
NORMAN J. LATKER,	REG. NO. 19,963	- IVER P. COOPER,	REG. NO. 28,005	- ALLEN C. YUN,	REG. NO. 37,971*
		- MANNY SCHECTER,	REG. NO. 31,722	- * Patent Agent	

ADDRESS ALL CORRESPONDENCE TO: **BROWDY AND NEIMARK, P.L.L.C.** 624 Ninth Street, N.W. Washington, D.C. 20001-5303	DIRECT ALL TELEPHONE CALLS TO: **BROWDY AND NEIMARK** (202) 628-5197

The undersigned hereby authorizes the U.S. Attorneys or Agents named herein to accept and follow instructions from <u>REINHOLD COHN AND PARTNERS</u> as to any action to be taken in the U.S. Patent and Trademark Office regarding this application without direct communication between the U.S. Attorney or Agent and the undersigned. In the event of a change of the persons from whom instructions may be taken, the U.S. Attorneys or Agents named herein will be so notified by the undersigned.

Page 2 of 2 **Atty. Docket:**

Title: COMPUTER-IMPLEMENTED METHOD AND SYSTEM FOR AUTOMATICALLY INVOKING A PREDETERMINED DEBUGGER COMMAND AT A DESIRED LOCATION OF A SINGLE THREAD OF A PROGRAM

U.S. Application Filed ______________, Serial No. __________

PCT. Application Filed ______________, Serial No. __________

I hereby further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief and believed to be true; and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

FULL NAME OF FIRST INVENTOR Eitan FARCHI	INVENTOR'S SIGNATURE [illegible]	DATE 27/7/2000
RESIDENCE 49 Hahashmonaim Street, Pardes Hanna 37052, Israel	CITIZENSHIP Israeli	
POST OFFICE ADDRESS 49 Hahashmonaim Street, Pardes Hanna 37052, Israel		
FULL NAME OF SECOND INVENTOR Shmuel UR	INVENTOR'S SIGNATURE Shmuel Ur	DATE 27/[illegible]/00
RESIDENCE Shorashim, D.N. Misgav 20164, Israel	CITIZENSHIP Israeli	
POST OFFICE ADDRESS Shorashim, D.N. Misgav 20164, Israel		
FULL NAME OF THIRD INVENTOR Avi ZIV	INVENTOR'S SIGNATURE [illegible]	DATE 27/7/2000
RESIDENCE 38 Leah Street, Haifa 34405, Israel	CITIZENSHIP Israeli	
POST OFFICE ADDRESS 38 Leah Street, Haifa 34405, Israel		
FULL NAME OF FOURTH INVENTOR	INVENTOR'S SIGNATURE	DATE
RESIDENCE	CITIZENSHIP	
POST OFFICE ADDRESS		
FULL NAME OF FIFTH INVENTOR	INVENTOR'S SIGNATURE	DATE
RESIDENCE	CITIZENSHIP	
POST OFFICE ADDRESS		
FULL NAME OF SIXTH INVENTOR	INVENTOR'S SIGNATURE	DATE
RESIDENCE	CITIZENSHIP	
POST OFFICE ADDRESS		
FULL NAME OF SEVENTH INVENTOR	INVENTOR'S SIGNATURE	DATE
RESIDENCE	CITIZENSHIP	
POST OFFICE ADDRESS		

Declaration